

Kurz und nützlich – Einzeiler

Einzeiler sind häufig sehr nützliche Utilities oder »Basic-Erweiterungen«. Wir stellen Ihnen deshalb hier einige dieser Kurzprogramme vor.

Wir wollen uns nicht nur auf's Listing beschränken, sondern Ihnen auch eine Beschreibung der Variablen und eine kurze Erklärung des Programmlaufs geben. So wird es sicher einfacher, die Kniffe, die in solchen Einzeilern stecken, zu verstehen und auch selbst zu verwenden.

Zahlenkonvertierungen von Dezimal nach Hexadezimal

Diese Zahlenkonvertierungen braucht man recht häufig.
– Hex X\$ nach dezimal (Listing 1)

```
– Hex X$ nach dezimal X:10
x=0:for i=1 to len(x$):
x0=asc(mid$(x$,i,1)):x=16*x
+x0-48+(x0>64)*7:next
```

und – Dezimal X nach hex X\$: (Listing 2)

```
– Dezimal X nach hex X$:10
x$="":for i=1 to 4:x0=x/16:
x=x-int(x0)*16:x$=chr$(
(48+x-(x>9)*7)+x0):next
x0:next
```

Umwandlung beliebiger Zahlensysteme

Die folgenden beiden Einzeiler von Martin und Hartmut Sprave dienen zum Umwandeln von Dezimalzahlen in Zahlen beliebiger Basis und umgekehrt. Man kann die beiden Programme auch kombinieren und erhält so eine Umwandlungsroutine zwischen verschiedenen Zahlensystemen. Beide sind auch als Unteroutine in einem Programm denkbar.

Zur Umwandlung dezimal/beliebig:

Die Variable D enthält eine Dezimalzahl beliebiger Größe. In der Variablen B muß die Basis angegeben werden, die der umgewandelten Zahl zugrunde liegt. Das Ergebnis steht in Z\$.

Zum Programm: Die Dummy-Schleife (von 0 bis 0) wird benutzt, um später wieder mitten in die Zeile springen zu können. An jeder Stelle wird die Zahl in der Variablen D durch die Basis B geteilt. Dadurch wird die unterste Ziffer abgeschnitten. Die jeweils niederwertigste Stelle ist der ganzzahlige Rest dieser Division und steht in S. Dieser wird in ASCII-Code umgerechnet. Durch den CHR\$-Befehl wird der Code zu einer Zeichenkette. Diese wird vorne an Z\$ angehängt. Die letzte höchstwertige Ziffer ist erreicht, wenn $D < 1$, da die nächste Stelle dann 0 ist. Solange $D > 1$ ist, ist die Endbedingung noch nicht erreicht und die FOR-Schleife wird weiter durchlaufen (bis $P+1 \leq 0$).

```
10 z$="":for p=0 to 0:d=d/b:s=(d-int(d))*b:
z$=chr$(55+s+7*(s<10))+z$:p=-d:next
40 rem in 10 dez in beliebig
```

Zur Umwandlung beliebig/dezimal:

Diese Routine wandelt eine beliebig große Zahl, deren Basis in der Variablen B steht, in eine Dezimalzahl um. Die Zahl selbst muß in Z\$ stehen. D wird mit 0 initialisiert. Die Schleifenvariable S dient als Zeiger auf die einzelnen Stellen von Z\$. Diese werden nacheinander in ASCII-Code umgewandelt. Der Code für Null (48) wird subtrahiert und das Ergebnis in H zwischengespeichert. Man multipliziert die umgewandelten Stellen (in D) mit B und erhöht sie dadurch um eine Potenz dieser Basis. Dann addiert man die aktuelle Stelle (H) dazu. Bei Darstellung durch einen Buchstaben ($H > 9$) ist aufgrund des ASCII-Codes noch die Subtraktion einer 7 notwendig. Die Schleife wird solange durchlaufen, bis die niederwertigste Stelle erreicht ist.

```
20 d=0:for s=1 to len(z$):h=asc(mid$(z$,s))-
48:d=d*b+h+7*(h>9):next
50 rem in 20 beliebig in dez
```

Die Fakultätsfunktion

Dieser Einzeiler von Detlev Marks berechnet Fakultäten besser als ein Taschenrechner (größer als »69«). Der C 64 besitzt standardmäßig keinen Befehl zur Berechnung der Fakultät.

Die Eingabevariable ist A. B dient als Zählvariable und C als Rechen- und Ausgabe-Variable.

```
10 rem fakultaeten
20 :
30 input a:frb=1 to a:c=c+log(b):next:c=c/
og(10):print 10^(c-int(c)):"e";int(c):run
40 :
```

Dividieren mit beliebig vielen Nachkommastellen

Die Variable Z enthält die Zahl, die geteilt wird, die Variable D den Dividenten, durch den geteilt wird. In N kann die Anzahl der gewünschten Nachkommastellen angegeben werden. Die Variable E% enthält das Ergebnis ohne Rest. Es wird ausgegeben. R enthält den Rest und in Q% steht der Quotient (beschränkt auf Vorkommastellen).

Die Vorkommastellen erhält man durch $Q = \text{INT}(Z/D)$, hier vereinfacht durch $Q\% = Z/D$. $Q\%$ wird ausgegeben. Dann wird der Rest $(Z - Q\% * D)$ berechnet. Die Schleife wird n mal durchlaufen ($N = \text{Anzahl der Nachkommastellen}$).

Analog zu $Q\%$ wird $E\% = R * 10/D$ berechnet. $E\%$ wird ausgegeben als 1. Stelle. Der neue Rest wird gebildet durch: $R(\text{neu}) = 10 * R(\text{alt}) - D * E\%$.

Nun kann die Schleife durchlaufen werden, bis alle Nachkommastellen ausgegeben wurden.

Beispiel: Wie lautet die 85. Nachkommastelle von $116/13$? $Z=116$, $D=13$, $N=85$ sind die Variablenwerte. Das Programm liefert das Ergebnis 9.

Der Autor dieses Programms ist Heinz Bauschke.

```
0 inputz,d,n;q%=z/d:printq%;r=z-d*q%;for
i=1ton:e%=r*10/d:print"||";e%;r=10*r-d*e
%:next
10 rem
```

INPUT mit Komma

Diese INPUT-Routine von Jürgen Reinert ersetzt den INPUT-Befehl. Sie erlaubt Komma, Doppelpunkt und Strichpunkt als zusätzliche Satzzeichen bei der Eingabe. Sonst funktioniert sie genauso wie der INPUT-Befehl. Die Routine übernimmt alle Zeichen der Tastatur, auch führende Leerzeichen (Leerzeichen vor Beginn des Textes).

Die Variable AA enthält die aktuelle Eingabe in ASCII-Code. II bildet die Laufvariable für Schleifen und XX\$ enthält den eingegebenen Text.

Das »Herz« dieses Einzelers ist die Eingaberoutine ab Adresse 42336. Diese schreibt alle 80 Zeichen einer Bildschirmzeile in den Basic-Eingabepuffer, der bei Adresse 512 beginnt. Dann liest das Programm Zeichen für Zeichen den Eingabepuffer bis zur genannten 0 und stellt daraus den String XX\$ zusammen.

Erfolgt keine Eingabe, das heißt, nur die RETURN-Taste wurde gedrückt, so wird die Routine mit XX\$=CHR\$(32) verlassen. In allen anderen Fällen enthält XX\$ alle sichtbaren, eingegebenen Zeichen (außer Steuerzeichen).

```
1 SYS 42336:XX$="":FOR II=512 TO 600:AA=PE
EK(II):IF AA THEN XX$=XX$+CHR$(AA):NEXT
10 :
20 REM INPUT
```

Formatierte Ausgabe

Dieser Einzeiler gibt Zahlen beliebiger Länge und unabhängig vom Vorzeichen rechtsbündig aus. Die Tabulatorfunktion von Volker Walter ist sicher beim Aufbau von Tabellen nützlich.

Die Variable A enthält die Zahl, die ausgegeben wird. Die Zahl »22« legt fest, daß die Kommata der ausgegebenen Zahlen in Spalte 24 stehen (2 Stellen werden vom Komma gebraucht). Die Spaltennummer kann im Listing beliebig gesetzt werden (Zahl im Listing = tatsächliche Spaltennummer - 2).

Ist $A > 1$, so wird der Zehnerlogarithmus von A berechnet und von der Spaltennummer abgezogen, bei der der Dezimalpunkt stehen soll. Da der C 64 nur den Logarithmus zur Basis e berechnen kann, muß das Ergebnis mit der Konstanten .43429448188 multipliziert werden. Dadurch erhält man den Zehnerlogarithmus.

Diese Berechnung erfolgt durch $\text{INT}(\text{LOG}(B-(B=0)) \cdot .43429448188) \cdot (B < 1)$.

Ist $A < 1$, so wird zur Spaltennummer eine Eins addiert. Null bildet einen Sonderfall, da die Null noch vor dem Komma stehen muß.

$\text{INT}(-B) \cdot (B < 1)$.

```
20 b=abs(a):printtab(int(log(b-(b=0))*.4
3429448188)*(b>=1)+int(-b)*(b<1)+22);a
```

Mehr Struktur mit Spaces

Der Basic-Interpreter auf dem C 64 überliest führende Leerzeichen zwischen Zeilennummer und dem Basic-Befehl einfach. So ist ein Einrücken mehrerer Programmzeilen oder das Einfügen von Leerzeilen, wodurch das Programm wesentlich übersichtlicher würde, standardmäßig nicht möglich.

In der Regel behilft man sich in solchen Fällen, indem man an den Zeilenanfang einen Doppelpunkt setzt, auf den dann die gewünschte Anzahl Leerzeilen folgt.

Von Herbert Heise stammt eine wesentlich elegantere Methode:

Der Basic-Interpreter überliest bei der Zeileneingabe Grafikzeichen. Diese Tatsache kann man sich zunutze machen. Nach der Zeilennummer wird irgendein Grafikzeichen (besonders einfach: SHIFT und gleichzeitig die Leertaste) getippt, danach die gewünschte Anzahl führender Leerzeichen und dahinter die vorgesehenen Basic-Befehle. Auf dem Bildschirm ist das Grafikzeichen noch zu sehen. Aber beim Auflisten stellt man fest, daß das Grafikzeichen nicht in den Programtext übernommen wurde. Dennoch erscheint die Zeile um die gewünschte Anzahl von Stellen eingerückt.

Um ganze Leerzeilen mit dieser Methode zu erzeugen, schreibt man hinter die Zeilennummer ein Grafikzeichen, ein Space und noch ein Grafikzeichen. Nach einem RETURN wird nur das Leerzeichen in die Zeile übernommen. Mit LIST erhält man nun eine Zeile, die nur aus einem Leerzeichen besteht.

In diesem Zusammenhang ist noch folgender POKE-Befehl interessant:

Mit »POKE 129,58« werden Basic-Zeilen nicht mehr ausgeführt, die keine Spaces in Anführungszeichen enthalten. Der Interpreter meldet »?SYNTAX ERROR«. Ausschalten läßt sich dieser Modus wieder mit »POKE 129,322«.

Zeilen löschen am Bildschirm

Diese kleine Routine von Stefan Keimeier löscht bestimmte Zeilen auf dem Bildschirm. Dabei wird eine Maschinenroutine des C 64 verwendet, die diejenige Zeile vom Bildschirm löscht, deren Zeilennummer im X-Register steht. Die Zeilennummer wird zunächst in die Speicherzelle gePOKEt, deren Inhalt der SYS-Befehl in das X-Register übernimmt. Dabei wird von 0 bis 24 gezählt. Die Position des Cursors bleibt davon unbeeinflusst.

Die Variable LN enthält die Zeilennummer (0 bis 24), V gibt die Von-Zeile und B die Bis-Zeile an (gelöscht wird »von« bis«).

```
10 FOR LN=V TO B:POKE 781,LN:SYS 59903:NEXT
20 :
30 REM FUER NUR EINE ZU LOESCHENDE ZEILEGILT:
40 :
50 POKE 780,LN:SYS 59903
60 :
70 REM
```

STEFAN KEIMEIER

Grafikbildschirm löschen mit DIM-Befehl

Die Variablen A und B müssen vor der DIM-Anweisung angelegt werden:

(A=0:B=0), da im Programmverlauf das Variablenfeld beeinflusst wird.

Dann werden durch

A=PEEK(49):B=PEEK(50) die Werte für das Variablenende gesichert. Durch das Dimensionieren einer Variablen wird ein entsprechend großer Platz hinter dem Variablenende auf Null gesetzt.

Der Bildschirmspeicher fängt bei Adresse 8192 und geht bis Adresse 16191. Hinter dem bisherigen Variablenende (Adresse: A+B*256) wird die Variable F (auf der Adresse 16191-a-b*256) angelegt. Pro indizierter Variablen werden 5 Byte freigemacht. Dabei wird die mit 0 indizierte Variable nicht berücksichtigt, ebenso die ersten sieben Byte für Variablenname und Dimension.

Durch POKE 49,A:POKE 50,B werden die alten Werte wieder hergestellt. F ist nun nicht mehr dimensioniert.

Wenn das Variablenende größer oder gleich 8186 ist, wenn also das Programm einschließlich Variablenfeld größer ist als 6 Bytes, dann wird der Bildschirm nicht ganz gelöscht.

Soll das Grafikprogramm zum Speichern von Bildern benutzt werden, darf man nicht vergessen, den Zeiger für Speichergrenzen (PEEK(55)+PEEK(56)*256) auf eine Adresse kurz hinter den Bildschirmspeicher zu setzen.

Der Autor des Programms, Manfred Hedtke, schlägt vor, im Grafikprogramm die Zeile 10 immer dann aufzurufen, wenn über die Tastatur »SHIFT + CTRL HOME« (entsprechend »freier Bildschirm im Textmodus«) eingegeben wird.

```
10 a=0:b=0:a=peek(49):b=peek(50):dimf((1
6191-a-b*256)/5):poke49,a:poke50,b
20 rem
```

Soft-Scrolling

Mit diesem Einzeiler von Georg Brandt kann ein beliebiger Text von rechts nach links punktweise über den Bildschirm geschoben werden. Die Variable A mit dem Wert 53270 bildet das Register für horizontales Smooth-Scrolling. Die Variable L enthält die Anzahl der Zeichen (L=40 für die gesamte Bildschirmbreite, weniger für kleinere Textausschnitte), die gleichzeitig auf dem Bildschirm erscheinen sollen. In der Variablen A\$ erwartet die Routine den zu zeigenden Text. Das letzte Zeichen sollte ein Leerzeichen sein. Das rechtsbündige Zeichen des Strings steht zweimal auf dem Bildschirm, da es nicht gelöscht wird.

Das Programm arbeitet nach folgendem Prinzip: Der Text wird auf den Bildschirm ausgegeben. Dann wird der Bildschirminhalt mit Hilfe des Smooth-Scrolling-Registers punktweise nach links gezogen, bis er um sieben Punkte verschoben ist. Nun wird der gesamte Text nach links geschoben und das Scroll-Register zurückgesetzt. Dadurch scheint der Text um den achten Punkt verschoben zu sein.

Wichtig:

- Vor Programmaufruf sollte der Bildschirm gelöscht werden, da sonst auch der restliche Bildschirminhalt verschoben wird.

- Auch andere Steuerzeichen außer dem hier verwendeten HOME können eingesetzt werden, um den Text zu positionieren.

```
1 FOR R=1 TO LEN(A$):FOR I=207 TO 200 STEP
-1:PRINT" (HOME)"MID$(A$,R,L):POKE A,I:NE
XT I,R
10 :
20 REM SCROLL
```

Scrollen in x-Richtung

Bei diesem Einzeiler von Hans-Peter Harmann kann die Geschwindigkeit des Scrollers je nach Anwendung neu eingestellt werden. Die Verzögerungsschleife

```
FOR G=0 TO 3:NEXT
```

muß nur entsprechend abgeändert werden. Durch den Wert 3 wird nur eine minimale Verzögerung erreicht. Experimentieren Sie ruhig mal damit, mit welchen Verzögerungswerten der Bildschirm wie schnell gescrollt wird.

```
10 f=0:t=1to7:poke53270,t:f=0to3:next:
next:onagoto10:fory=1024to2023:pokey,194
:next:a=1:goto10
20 rem
```

Scrollen bei bleibendem Text

Dieses kleine Programm stammt von Peter Eckart. Es bewirkt, daß nur Teile des Bildschirms gescrollt werden. Die Zahl, die Sie in Speicherzelle 59639 POKEn, legt fest, wieviele Zeilen am oberen Rand stehen bleiben. So kann man die Kopfzeile einer Tabelle auf dem Bildschirm festhalten oder eine Information, die für ein Programm wichtig ist (Tastenbelegung, Erklärung der wichtigsten Befehle oder anderes). Beendet wird dieses Programm durch den Befehl POKE 1,55.

Wenn Sie eine »0« gePOKEt haben, wird eine Zeile auf dem Bildschirm festgehalten. Beim POKEn einer »1« zwei Zeilen, beim POKEn einer »3« ...

Obwohl diese Routine sehr kurz ist, braucht sie sehr lange zur Ausführung.

```
1 for i=40960to49151:pokei,peek(i):next:f
ori=57344to65535:pokei,peek(i):next:poke
59639,10:poke1,53
```

Grafik: Rahmen zweifarbig

Der Bildrahmen des C 64 ist standardmäßig einfarbig. Von Markus Hillebrand stammt dieser Einzeiler, der einen zweifarbenen Außenrahmen simuliert.

So einfach diese Routine ist: sie verblüfft einen doch!

Der Rahmen erscheint in schwarz (0) und weiß (1). Beim Abtippen der Programmzeile ist es ganz wichtig, die Anzahl der Doppelpunkte und der Leerstellen genau zu treffen. Zwischen den beiden POKEs und den Doppelpunkten stehen je fünf Leerzeichen. Danach folgen 17 beziehungsweise 15 Doppelpunkte. Wichtig ist dies deshalb: Der Autor des Programms wollte die Grenze zwischen den beiden Farben möglichst ruhig auf dem Bildschirm haben. Dazu mußte er die genaue Farbwechselperiode finden. Da der Interpreter Doppelpunkte und Leerzeichen unterschiedlich schnell abarbeitet, hat er die Feineinstellung mit Leerzeichen vorgenommen.

Aber Vorsicht: Sobald während des Programmlaufs eine Taste gedrückt wird, fängt das Bild an zu »laufen«, weil die Raster-Interrupt-Tastaturabfrage in Kraft getreten ist, die den Zeitplan durcheinander bringt.

```
10 poke53280,1 .....:poke
53280,0 .....:goto10
20 rem
```

Spiralen mit dem Plotter 1520

Dieser Einzeiler malt mit dem Plotter 1520 eine Spirale, die aus lauter Dreiecken besteht. Zugrunde liegt diesem Programm von Christoph von Rhein die Berechnung eines Kreises. Die X- und Y-Koordinaten werden mit den Funktionen SIN und COS errechnet. Da X und Y mit der Laufvariablen I multipliziert werden, kommt bei dieser Berechnung eine Spirale heraus. Wenn Sie das Aussehen der Grafik ändern wollen, dann können Sie den Faktor 2 innerhalb der Sinus- und Cosinus-Klammern durch einen anderen Wert ersetzen.

```
1 OPEN 1,6,1:PRINT#1,"M",240,0:FOR I=1 TO
250:PRINT#1,"S",240+SIN(I*2)*I,COS(I*2)*
I:NEXT
```

Ein grafischer Disassembler: DI-AS

Der Speicherbereich des C 64 läßt sich in 256 Seiten à 256 Byte aufteilen. DI-AS interpretiert die Speicherinhalte als Bild-

schirmcode und stellt die 64 KByte des C 64 Seite für Seite auf den Zeilen 7-13 des Bildschirms dar. Die Seitennummer wird am oberen Bildschirmrand angezeigt. Andreas Carl, der Autor dieses Programms, sagt zur Bedienung des Dissassemblers:

Bevor das Programm geladen und gestartet wird, muß mit »SYS64738« unbedingt ein Reset durchgeführt werden. Dann sollte der Bildschirm mit »CLR« gelöscht werden. Mit »RUN« wird das Programm gestartet. Dann stehen dem Benutzer die folgenden Optionen offen:

- im Speicher vorwärts blättern (CURSOR-RIGHT)
 - im Speicher rückwärts blättern (CURSOR-DOWN)
 - die Seite NR betrachten (R/S POKE3,NR:CLR:RUN)
- Es stehen umfangreiche Fehlerbehandlungen zur Verfügung.

```
5 print "13b3q3(HP- : "peek(3) " " : pok
e41,5: geta$:poke3,peek(3)-(a$=" ")+ (a$="
3") : sys1024: goto5
```

Speicherblockverschiebung

Dieser Einzeiler von Jens Baas dient zur Übertragung von Speicherblöcken. Er ist zum Beispiel nützlich, wenn Sie das Basic oder das Betriebssystem vom ROM ins RAM verlegen wollen. Die Variablen mit »L« bezeichnen jeweils das Low-Byte der Adresse, die Variablen mit »H« das High-Byte. Für eine Adresse X lassen sich diese so berechnen: AL=X-256*INT(X/256):AH=INT(X/256). Die Routine benutzt die Blockverschiebe-Routine des ROMs.

```
1 poke95,al: poke96,ah: poke90,el: poke91,e
h: poke88,nl: poke89,nh: sys41919
11 rem beispiel:
12 poke95,0: poke96,160: poke90,0: poke91,1
92: poke88,0: poke89,192: sys41919
13 rem ↑↑ basic ins ram ↑↑
```

Merge

Bisher gab es oft nur die eine Möglichkeit, wenn ein Programm aus zweien zusammengesetzt werden sollte: das kürzere mußte ans Ende des längeren getippt werden. Hier ist eine ganz kurze Routine von Andreas Gast, die Programme verkettet.

```
10 a=peek(45)+256*peek(46)-2: poke44,a/25
6: poke43,a-peek(44)*256: print "prg laden
& p043,1: p044,8
```

Directory laden, ohne das Basic-Programm zu zerstören

Jedem Programmierer ist das Problem, das diesem Einzeiler zugrunde liegt, sehr bekannt: Man hat - mit Ach und Krach - ein Basic-Programm geschrieben. Nun will man seine Arbeit speichern und erkennt seine freien Disketten nicht wieder. Wenn man die Disketten aber mit LOAD "\$",8 : LIST

listet, löscht man das neue Programm und die ganze Arbeit war umsonst. Reinhard Abdel-Hamid schlägt vor, die folgende Programmzeile vor dem eigenen Programm einzugeben. Mit ihr läßt sich das Directory listen. Das im Speicher liegende eigene Programm bleibt erhalten.

Gestartet wird mit
OPEN 1,8,2,"\$":GOTO 0.

Der OPEN-Befehl eröffnet eine sequentielle Datei (hier das Directory) zum Lesen. Mit

GET #1,A\$

wird ein Byte vom Disketten-Puffer geholt und das Kommando

A=ASC(A\$+"SHIFT/HOME")

wandelt ASCII-Zeichen in Zahlen (von 0 bis 255) um. Hat A\$ die Länge 0, so behält der Ausdruck trotzdem die Länge 1. Durch

PRINT CHR\$((A=130 AND 13 OR ((31<A AND A<95) AND A)));

wird das zugehörige Zeichen zu A gedruckt, wenn A einen Wert zwischen 32 und 95 hat. Ist A=130, so wird RETURN ausgegeben. Die Steuerzeichen und Grafiksymbbole werden durch die Formel innerhalb der Charakterstring-Klammer »herausgefiltert«. Nur Zahlen, Buchstaben und Satzzeichen werden ausgedruckt.

```
0 get#1,a$:a=asc(a$+"SHIFT/HOME"):printchr$((a=13
0and13or((31<aanda<95)anda)));:goto0
```

Die Routine kann auch für andere Zwecke angewendet werden.

Es ist ohne weiteres möglich, alle Kommentare, Inhalte von Print-Anweisungen und Texte des auf der Diskette befindlichen Programms auf den Bildschirm zu bringen, da man alle Programme auf der Diskette als sequentielle Datei lesen kann.

Eröffnen Sie die Datei dafür nach dem folgenden Schema:
OPEN 1,8,2,"filename"

Dann wird das Programm mit

GOTO 0 gestartet. Wenn die Ausgabe beendet ist, wird mit der RUN/STOP-Taste die Endlosschleife wieder verlassen.

Maschinenprogramme speichern

Mit dem folgenden Einzeiler von Markus Eicher können Maschinenprogramme sehr einfach gespeichert werden, ohne den Basic-Pointer zu verstellen. Die Werte für LE und HE werden berechnet, indem man auf die Endadresse eine 1 addiert.

Le = Low-Byte der Endadresse

He = High-Byte der Endadresse

La = Low-Byte der Anfangsadresse

Ha = High-Byte der Anfangsadresse

Low- und High-Byte einer Adresse werden so berechnet:

Adresse = dez. 2000

Highbyte = INT(2000/256)=78

Lowbyte = 20000-Lowbyte*256=32

Die Variable A\$ enthält den Namen der Datei, in die auf der Diskette das Maschinen-Programm abgelegt werden soll. X legt fest, ob das Programm auf Diskette (X=8) oder Kassette (X=1) gespeichert wird.

Falls ein langer Programmname gewählt wird, müssen die Befehle abgekürzt werden.

```
1 sys(57812)a$,x: poke193,ls: poke194,hs: p
oke174,le: poke175,he: sys62957
```

Beispiel 1:

Das Programm auf den Adressen 20000 bis 22000 soll auf der Diskette unter dem Namen »Beispiel1« gespeichert werden. Der Dateiname wird der Variablen \$A zugewiesen. Die POKE-Befehle lauten

POKE193,32:POKE194,78;POKE174,241:POKE175,85

Beispiel 2:

Ein Programm von \$C000-\$C37E soll auf Kassette gespeichert werden. Die Variable X hat also den Wert 1.

Die Low- und High-Bytes sind:

1. von \$C000

Low = \$00 = dez. 0

High = \$C0 = dez. 192

2. von \$C37E

Low = \$7E = dez. 126

High = \$C3 = dez. 195

Die POKE-Befehle heißen:

POKE193,0:POKE194,192:POKE174,127:POKE175,195

Floppy-Zeit verkürzen

Die kleine Routine von Robert Loos dient dazu, die Zugriffszeit der Floppy-Disk 1541 ganz entscheidend zu verkürzen. Eine sichere Funktion der Floppy wird nicht gefährdet, wenn der Schrittmotor, der den Schreib-Lesekopf bewegt, wesentlich schneller arbeitet. Der Schrittmotor wird im Interrupt bedient. Daher genügt es, die Größe des Interruptintervalls zu verändern, um die Drehzahl des Motors zu beeinflussen. Standardmäßig wird etwa alle 15 Millisekunden (hier etwa alle 4 Millisekunden) ein Interrupt ausgelöst, der den Stepper um eine Viertelspur bewegt. Alle Bewegungen des Kopfes werden dadurch fast viermal schneller. Vorteile sind neben der Zeitersparnis:

Das Laufgeräusch wird angenehm leise und kurz, und im Falle einer Kopfjustage fährt der Kopf mit erheblich verminderter Kraft gegen den Anschlag, sodaß die Gefahr einer Dejustage deutlich gemindert ist.

```
10 OPEN 1,8,15,"M-W"+CHR$(7)+CHR$(28)+CHR$(1)+CHR$(15)
20 :
30 REM ZUGRIFFSZEIT DER FLOPPY KUERZER
```

Ein RENEW, das funktioniert!

Noch ein Programm von Hartmut und Martin Sprave: Dieser Einzeiler ist im Direktmodus, also ohne Zeilennummern, einzugeben. Vorher dürfen keine Variablen definiert oder Basic-Zeilen eingetippt werden, da sonst das gelöschte, aber noch im Speicher befindliche Programm zerstört wird. Im Wesentlichen besteht das RENEW-Programm auf dem Aufruf einer System-Routine, die die Basic-Zeilen neu bindet und das Ende des gelöschten Programms herausfindet. Das erste POKE tut so, als ob sich ein (ungelöschtes) Basic-Programm im Speicher befindet, weil sonst die Routine nicht arbeitet. Die Endadresse des Basic-Programms wird um zwei erhöht und in den Zeiger auf den Start der Variablen (45/46) übertragen. Der CTR-Befehl gleicht alle weiteren Basic-Zeiger diesem Befehl an.

```
6 rem c-64
7 poke2050,8:sys42291:poke46,peek(35)-(peek(781)>253):poke45,peek(781)+2and255:c1r
```

Ein einfaches Renumber

Diese Routine von Georg Wichert numeriert ein Basic-Programm, das bis zu 255 Zeilen hat, in wenigen Sekunden neu. Die erste Zeilennummer ist dann 0, die Schrittweite ist 1, Sprungadressen der Befehle GOTO und GOSUB bleiben

beim Neunumerieren unverändert. Jedes Basic-Programm belegt die Speicherplätze 2048 bis PEEK(45)+PEEK(46)*256-3. PEEK(2049)+PEEK(2050)*256 gibt an, bei welcher Adresse die erste Zeile aufhört. Die erste Zeilennummer A wird aufgrund der Adressen 2051 und 2052 folgendermaßen berechnet:

$A = \text{PEEK}(2051) + \text{PEEK}(2052) * 256$.

Die zweite Zeilennummer findet man im Speicher an den Adressen

$\text{PEEK}(2049 + \text{PEEK}(2050) * 256 + 2)$ und

$\text{PEEK}(2049 + \text{PEEK}(2050) * 256 + 3)$.

Die Zeilennummern befinden sich jeweils an den Adressen A+2 (Low Byte) und A+3 (High Byte).

Ist das Basic-Programm, das neu nummeriert wird, länger als 255 Zeilen, dann muß der Einzeiler zum Zweizeiler abgeändert werden. Dann wird folgendes wichtig: Aus Platzgründen wird in dieser Programmversion an die Stelle A+2 der Wert z gePOKET (z=0,1,2,...,n-1; n-1 steht für die Anzahl der Zeilen des Basic-Programms, das umnummeriert werden soll, inklusive dem Einzeiler) und an die Stelle A+3 den Wert 0.

```
1 FOR A=2049 TO PEEK(45)+PEEK(46)*256-3:PO
KE A+2,Z:POKE A+3,0:A=PEEK(A)+PEEK(A+1)*
256-1:Z=Z+1:NEXT
10 :
20 REM RENUMBER (GEORG WICHERT)
```

Track-Zerstörer: Kopierschutz

Diese Programmzeile von Jörg Wegmeyer produziert auf dem gewünschten Track der Diskette (Variable T) einen READ ERROR 21. Damit erreicht man einen relativ sicheren Kopierschutz.

```
1 open1,8,15:open2,8,2,"#":print#1,"u1 2
0";t;0:print#1,"m-e"chr$(163)chr$(253)
```

Eine Zeile = kompletter Datenschutz

Diese Routine von Volker Ritzhaupt erreicht, daß auf einer Diskette weder das Inhaltsverzeichnis gelistet, Programme gelöscht noch irgendetwas auf die Diskette geschrieben werden kann. Das Laden der Programme funktioniert hingegen ganz normal. Der Inhalt der Diskette kann nur durch Formatieren mit einer neuen ID-Nummer noch manipuliert werden.

Das Directory wird nicht mehr gelistet, weil es als Basic-Programm geladen wird und in dem veränderten Directory drei Nullen am Anfang (für den Interpret der Zeichen für Programmende) erscheinen. Das »Directory-Programm« endet also schon nach zehn Bytes. Die ersten fünf Bytes stellen den Zeilenanfang und die Zeilennummer dar. Darauf folgen, wie immer am Anfang eines Inhaltsverzeichnisses, ein Leerzeichen und ein Anführungszeichen. Um diese drei Zeichen beim Listen verschwinden zu lassen, folgen nun drei chr\$(20), also drei DELETES. Dies bewirkt, daß das Directory beim Listen völlig verschwindet. Da nun unmittelbar die drei Nullen folgen, wird das Listen hier abgebrochen. Auf den LIST-Befehl erscheint nur die Meldung »READY«.

```
1 open1,8,3,"#":open2,8,15,"b-p3,144":pr
int#1,"###"chr$(0)chr$(0)chr$(0):print#2
,"u2:3,0,18":print#2,"i
2 rem
3 rem verhindert auflisten des directory
und macht schreibschutz auf disk
4 rem
```

Der Befehl GOTO X - eine Basic-Erweiterung

Dieses Programm von Reinhard Jurk simuliert den im Commodore-Basic nicht vorhandenen Befehl GOTO X.

Zu dem etwas seltsamen Aussehen der Programmzeile: In den Ausführungszeichen steht ein Maschinenprogramm, dessen Opcodes im Listing des Basic-Interpreters diese merkwürdigen Zeichen erzeugen.

```
10 syspeek(61)+256*peek(62)+26: "L=efl ll
u=V H0u:JH0u - - - - - +_ "1"
20 rem
30 rem goto x: 11%=zeilennummer:goto10
40 rem
```

BITTE UNBEDINGT GENAU SO EINGEBEN !!!
43 KLAMMERAFFEN (@)

```
5 GOTO20
10 SYSPEEK(61)+256*PEEK(62)+26: "oooooooo
oooooooooooooooooooooooooooooooooooo"
20 FORA=2082TO2124:READX:S=S+X:POKEA,X:N
EXT:IFS<>5185THENPRINT"DATA ERROR !":END
30 PRINT"OK."
40 DATA169,204,133,69,133,70,165,157,133
,85,133,86,160,12,200,145,85,162,129
50 DATA202,200,138,145,85,172,128,163,16
6,46,165,45,32,17,177,32,43,175,32,1
60 DATA184,76,163,168
READY.
```

Der Einzeler als DATA-Lader

```
Cx
PC. IRQ NU-BDIZC AC XR YR SP
.; E147 EA31 00110001 08 08 02 F6

., 081A A9 CC LDA #CC
., 081C 85 45 STA $45
., 081E 85 46 STA $46
., 0820 A5 9D LDA $9D
., 0822 85 55 STA $55
., 0824 85 56 STA $56
., 0826 A0 0C LDY #$0C
., 0828 C8 INY
., 0829 91 55 STA ($55),Y
., 082B A2 81 LDX #$81
., 082D CA DEX
., 082E C8 INY
., 082F 8A TXA
., 0830 91 55 STA ($55),Y
., 0832 AC 80 A3 LDY $A380
., 0835 A6 2E LDX $2E
., 0837 A5 2D LDA $2D
., 0839 20 11 B1 JSR $B111
., 083C 20 2B AF JSR $AF2B
., 083F 20 01 B8 JSR $B801
., 0842 4C A3 A8 JMP $A8A3
```

Die Bedeutung des DATAs im Assembler-Code

Zur Eingabe:

In diesem Listing ist ein Generierungsprogramm angegeben. Dieses wird eingegeben und gestartet. Wenn der Checksummer-Test positiv ausgefallen ist und man sich vergewissert hat, daß das Programm richtig eingegeben wurde, gibt man »LIST« ein, gefolgt von »NEW«. Dann fährt man mit dem Cursor auf Zeile 10 und drückt RETURN. Die Routine steht nun im Speicher.

Zum Einbinden der Routine in eigene Programme ändert man die Zeilennummern entsprechend ab, weist der Variablen LL% den Wert der anzuspringenden Zeile zu und springt mit »GOTO« zu der Zeile, in der GOTO X steht.

File-Einträge mit Sonderzeichen

Mit dem Einzeler von Ralf Peiler lassen sich Zusätze an Filenamen im Directory anhängen. So wird zum Beispiel aus dem Directory-Eintrag "name" der Directory-Eintrag "name",8: oder "name",8,1. Doch nun der Einzeler:

```
INPUT" name";A$:OPEN 1,8,15, "R:"+A$+"{SHIFT-SPACE}
{COMMODORE-D}8{SHIFT-KLAMMERAFFE}="+A$:CLOSE 1
»SHIFT-SPACE« teilt dem DOS mit, ein Anführungszeichen zu setzen.
```

»COMMODORE-D« ist der Code für »,« »8« entspricht der normalen »8« und »SHIFT-KLAMMERAFFE« ist der Code für »,«. Soll »,8,1« an den Filenamen angehängt werden, so ist »[COMMODORE-D]8[SHIFT-KLAMMERAFFE]« zu ersetzen durch »[COMMODORE-D]8[COMMODORE-D]«. Wie Sie sehen, geht das Anhängen kinderleicht. Verblüffung kann man auch erzeugen durch ein buntes Directory. Allerdings lassen sich nicht sämtliche Farben erzeugen. Bisher sind folgende Kombinationen gefunden worden:

```
name {SHIFT-SPACE} {CTRL-2} = name → weiß
name {SHIFT-SPACE} {CTRL-3} = name → rot
name {SHIFT-SPACE} {CTRL-4} = name → blau
name {SHIFT-SPACE} {CTRL-6} = name → grün
name {SHIFT-SPACE} {CTRL} = name → clr
name {SHIFT-SPACE} {DEL} = name → del (auch mehrmals)
name {SHIFT-SPACE} {RVSON} = name → bis »PRG« revers
```

Trick 17 mit ON ... GOTO

Eine sehr interessante Version einer ON...GOTO-Anweisung stammt von Peter Zankl.

Der Ausdruck (A\$="A") hat den Wert -1, wenn ein A eingegeben wurde. Sonst hat er den Wert 0. Rechnen Sie nach oder probieren Sie es aus! (cg)

```
100 rem tastaturabfrage mit sprung
200 rem :
300 rem vorher:
400 :
410 geta$:ifa$=""then410
420 ifa$="a"then2000:rem programmteil a
430 ifa$="b"then3000:rem programmteil b
440 ifa$="x"then end:rem ende
450 goto 410
499 :
500 rem nachher:
600 :
610 geta$:on1-(a$="a")-2*(a$="b")-3*(a$="x")goto610,2000,3000:end
620 :
```